Kata de Arquitectura diseñada para evaluar los conocimientos de Arquitectura Empresarial, Liderazgo técnico y Arquitectura de Soluciones de Flypass, alineada con la metodología **Why Driven Design (WhyDD)** usada por flypass para diseño arquitectónico.

Kata de Arquitectura "FinScale Evolution"

1. Introducción y Rol

Usted ha sido contratado como el **Arquitecto Principal (Enterprise & Solution)** para "FinScale", una Fintech en hipercrecimiento. Su rol requiere una combinación de visión estratégica (Arquitectura Empresarial), profundidad técnica (Arquitectura de Soluciones) y capacidad de mentoría (Liderazgo Técnico).

Se espera que utilice la metodología **Why Driven Design (WhyDD)** para estructurar su solución, justificando cada decisión desde el negocio hasta la implementación técnica.

2. El Escenario de Negocio (Contexto)

La Empresa: FinScale nació hace 15 años procesando pagos locales. Hoy, opera en 12 países y está lanzando "GlobalLedger", una plataforma de gestión financiera global en tiempo real y detección de fraude.

El Problema Actual: Todo el sistema corre sobre un Monolito en Java 8 (J2EE) con una base de datos Oracle centralizada.

- Dolor de Negocio: El Time-to-market es de 4 meses. Cada despliegue requiere una ventana de mantenimiento de 6 horas.
- Dolor Técnico: Bloqueos en base de datos constantes. Imposibilidad de escalar módulos individualmente (ej. el módulo de fraude tumba el de pagos).
- El Reto de Escala: Se espera pasar de 2,000 TPS (Transacciones por Segundo) a picos de 1,000,000 TPS en eventos masivos de mercadeo, manteniendo consistencia estricta en saldos y eventual en reportes.

Objetivos Estratégicos:

1. Escalabilidad Extrema: Soportar millones de TPS con arquitectura reactiva.
2. Disponibilidad del 99.999%. El sistema no puede caerse completo nunca.
3. Modernización: Migrar a Java Spring Boot (Stack Reactivo/WebFlux) y Arquitectura Cloud Native.
4. Gobierno de Datos: Cumplimiento estricto de PCI-DSS y GDPR, con trazabilidad total.
5. Resiliencia, un fallos catastrófico en una parte del ecosistema no debe afectar el procesamiento de pagos.

Escuela de Arquitectura Tecnológica.

Funcionalidades requeridas.

A. Equipo: "Growth & Customer Experience" (Canales y Cliente)

Este equipo es responsable de la adquisición de usuarios y la interacción directa a través de la App Móvil y Web.

Funcionalidad	Descripción	Datos Requeridos (Input/Output)
Customer Onboarding	Registro de nuevos usuarios (personas y empresas), captura de biometría y documentos legales.	Fotos de ID, Selfies biométricos, Datos fiscales (RUT/TaxID), Dirección física, Estructura accionaria (para B2B).
Account Access (Authentication)	Gestión de credenciales, 2FA, recuperación de claves y manejo de sesiones.	Tokens OAuth2, Hashes de password, Device Fingerprint, OTPs, Historial de logins.
Wallet Overview (Current Account)	Visualización de saldos consolidados en múltiples divisas y movimientos históricos.	ID de Cliente, Saldos por divisa, Lista de transacciones (metadata simple), Estado de cuenta.
Beneficiary Management (Party Data Management)	Alta y baja de terceros frecuentes para transferencias.	Alias, IBAN/SWIFT, Nombre del banco destino, Tipo de relación.

B. Equipo: "Global Operations" (Tesorería y Procesamiento)

Es el corazón transaccional. Se encarga de que el dinero se mueva y se concilie.

Funcionalidad	Descripción	Datos Requeridos (Input/Output)
Payment Execution (Payment Order)	Orquestación del pago: validación de saldo, reserva de fondos y enrutamiento a la red de salida adecuada para procesar el pago.	Monto, Divisa origen/destino, ID Ordenante, ID Beneficiario, Fecha valor, Metadata de referencia.

FX Trading (Currency Exchange)	Cotización y ejecución de cambio de divisas en tiempo real para pagos transfronterizos.	Pares de divisas (ej. USD/EUR), Spread comercial, Tasa del mercado (Spot rate), Timestamp de bloqueo de tasa.
Clearing & Settlement (Correspondent Bank Ops)	Comunicación con redes bancarias externas (SWIFT, SEPA, ACH) para liquidar los fondos.	Mensajes ISO 20022 (pacs.008, pacs.002), Archivos de lote (Batch files), ACKs de la red.
Liquidity Management (Treasury)	Monitoreo de las cuentas de FinScale en bancos partner para asegurar fondos suficientes.	Saldos en bancos corresponsales, Pronósticos de cash-flow intradía, Alertas de descubierto.

C. Equipo: "Trust & Safety" (Riesgo y Cumplimiento)

Encargados de proteger la plataforma y cumplir con regulaciones globales.

Funcionalidad	Descripción	Datos Requeridos (Input/Output)
Fraud Detection (Fraud Resolution)	Análisis en tiempo real de cada transacción para detectar patrones anómalos o account takeover.	Velocidad de transacción, Geolocalización IP vs GPS, Historial de comportamiento, Reglas de score.
Sanctions Screening (Watchlist Screening)	Verificación de que ni el ordenante ni el beneficiario estén en listas negras (OFAC, Interpol).	Nombres completos, Fechas de nacimiento, Países de riesgo, Listas PEP (Personas Políticamente Expuestas).
Regulatory Reporting (Regulatory Compliance)	Generación de reportes para bancos centrales y entidades fiscales.	Logs de transacciones enriquecidos, Trazabilidad de fondos, Reportes de operaciones sospechosas (ROS).

D. Equipo: "System of Record" (Contabilidad Técnica)

El área encargada de la verdad financiera de la empresa.

Funcionalidad	Descripción	Datos Requeridos (Input/Output)
Position Keeping (Financial Accounting)	El "Ledger". Registro inmutable de doble entrada de cada movimiento de dinero.	ID Transacción, Cuenta Debe, Cuenta Haber, Monto, Timestamp atómico.
Reconciliation (Reconciliation)	Comparación automática entre el Ledger interno y los extractos bancarios de los aliados.	Archivos MT940/CAMT.053 (extractos bancarios), Registros internos del Ledger.

Aliados e Integraciones (Ecosistema)

FinScale Evolution no opera de manera aislada. El candidato debe considerar estas restricciones de integración en la etapa de "Entender el contexto y las restricciones técnicas"

- Esquemas de Tarjetas (Visa/Mastercard):
  - Integración: ISO 8583 (Legacy, sockets TCP) y APIs REST modernas.
  - Requisito: Latencia < 200ms para autorizaciones.
- Redes de Compensación Local (ACH, SPEI, SEPA, PIX):
  - Integración: Varía por país. Archivos Batch (SFTP) para ACH, APIs REST/SOAP para inmediatos (PIX/SPEI).
  - Requisito: Alta disponibilidad. Si SEPA cae, FinScale debe encolar y reintentar.
- Proveedores de KYC/AML (Jumio, Onfido, World-Check):
  - Integración: Webhooks asíncronos.
  - Flujo: Se envía la foto del ID, el proveedor responde minutos después con el veredicto.
- Bancos Corresponsales (JP Morgan, Citi, BBVA):
  - Integración: Conexión Host-to-Host (H2H) para transmisión de archivos seguros y APIs de Open Banking.

Procesos de Negocio Críticos (Para Domain Storytelling)

Estos procesos deben ser utilizados por el arquitecto para realizar la técnica de Domain Storytelling sugerida en WhyDD.

Proceso 1: Transferencia Internacional (P2P) - "El Camino Feliz"

1. Cliente inicia transferencia desde la App (USD a EUR).

2. Sistema cotiza la tasa de cambio (FX) y la bloquea por 5 minutos.
3. Cliente confirma con biometría.
4. Motor de Fraude analiza en < 100ms. Si es verde -> continua.
5. Ledger debita saldo del cliente (USD) y acredita cuenta puente interna (USD).
6. Tesorería instruye al Banco Partner en Europa para pagar al destinatario final (EUR).
7. Sistema notifica al cliente "Envío en proceso".
8. Al recibir confirmación del Banco Partner, Ledger mueve fondos de cuenta puente a cuenta liquidada.

Proceso 2: Reconciliación (Batch Nocturno) - "El Dolor de Cabeza"

1. A las 00:00 UTC, FinScale descarga los extractos de todos los bancos aliados.
2. El sistema cruza cada movimiento bancario contra el Ledger interno.
3. Desviaciones: Si el banco dice que salieron $100 pero el Ledger dice $90, se genera una alerta de nivel crítico para el equipo de operaciones manuales. Actualmente este proceso tarda 6 horas y bloquea la base de datos.

Deuda técnica.

El sistema actual ("LegacyCore") es un nudo gordiano de dependencias técnicas que se deben tener en cuenta en el diseño sin detener la operación del negocio (que mueve millones de dólares diarios).

A. Datos y Persistencia (El mayor bloqueante)

1. Lógica de Negocio en Base de Datos (PL/SQL):
    - Aproximadamente el 40% de las reglas de negocio críticas (cálculo de comisiones, validación de saldo, bloqueos de riesgo) no están en el código Java, sino en Store Procedures de Oracle (PL/SQL) complejos y anidados.
    - Dificultad: No se puede simplemente "extraer el código Java" a un microservicio. Migrar implica reescribir lógica de PL/SQL a Java, con el riesgo de inconsistencia en el comportamiento.
2. Integración por Base de Datos Compartida (Shared Database):
    - Sistemas satélites (Reportes, Fraude Legacy, CRM antiguo) no consumen APIs, sino que leen y escriben directamente en las tablas del CORE_SCHEMA.
    - Dificultad: Cambiar el esquema de la base de datos para el nuevo sistema romperá integraciones desconocidas. Se requiere una estrategia de Database Refactoring o vistas de compatibilidad.
3. Secuencias y IDs Globales:
    - El sistema depende de secuencias de Oracle centralizadas para generar IDs de transacción únicos y consecutivos.
    - Dificultad: En un sistema distribuido/microservicios, generar IDs secuenciales centralizados es un cuello de botella de rendimiento y punto único de fallo.

B. Conectividad y Protocolos

4. Protocolos Stateful y Legacy (ISO 8583 sobre TCP crudo):
    - La conexión con las redes de tarjetas (Visa/Mastercard) utiliza sockets TCP persistentes y el protocolo binario ISO 8583.
    - Dificultad: Los microservicios modernos en Kubernetes son efímeros y stateless. Mantener conexiones TCP persistentes y estado de sesión en un entorno de contenedores elástico es complejo y requiere un diseño cuidadoso (Sidecars, Proxies dedicados).
5. Dependencia de APIs Síncronas Bloqueantes:
    - Los clientes actuales (App móvil antigua, Web legacy) esperan una respuesta HTTP síncrona inmediata (200 OK) tras enviar una transacción.
    - Dificultad: La nueva arquitectura es Event-Driven (Asíncrona). El arquitecto debe diseñar cómo puentear un frontend que espera respuesta inmediata con un backend que procesa en segundo plano (Polling, WebSockets, o "Fake Synchronous").

C. Infraestructura y Seguridad

6. Dependencia de Hardware Físico (HSM):
    - Las llaves criptográficas para firmar transacciones y cifrar datos sensibles (PINs) residen en HSMs (Hardware Security Modules) físicos en un datacenter on-premise.
    - Dificultad: La migración a la Nube (AWS/Azure) debe resolver cómo acceder a estos HSMs físicos con latencia mínima, o cómo migrar a Cloud HSMs sin interrumpir el servicio.
7. Ventana de Batch Nocturno (The "Batch Window"):
    - El sistema se apaga o se bloquea (Read-Only) todos los días de 02:00 AM a 04:00 AM para correr procesos de cierre y conciliación masiva.
    - Dificultad: El negocio exige ahora operación 24/7. La nueva arquitectura debe eliminar la necesidad de ventanas de mantenimiento, procesando cierres "al vuelo" sin bloquear la base de datos.

D. Código y Despliegue

8. Sesiones de Usuario en Memoria (Sticky Sessions):
    - El monolito Java guarda el estado de la sesión del usuario en la memoria RAM del servidor (HttpSession).
    - Dificultad: Impide el escalado horizontal dinámico. Si un servidor muere, se desconecta a todos sus usuarios. La migración debe externalizar el estado (ej. a Redis), pero refactorizar el código legacy para esto es costoso.
9. God Classes y Acoplamiento Cíclico:
    - Existe una clase TransactionManager.java con más de 15,000 líneas de código que es importada por casi todos los módulos (Pagos, Clientes, Notificaciones).

- Dificultad: Es casi imposible compilar o desplegar el módulo de "Notificaciones" sin desplegar también el de "Pagos". Esto dificulta la estrategia de Strangler Fig por componentes aislados.

3. El Desafío (Entregables Esperados)

Debe presentar un documento de diseño de arquitectura que cubra las siguientes secciones, siguiendo el flujo de **Why Driven Design**.

Etapa 1: Definición del Reto y Entendimiento del Negocio

Evaluación de: Arquitectura Empresarial, Business Capabilities, WhyDD.

1. Contexto de Negocio:
   - Utilice la técnica de Domain Storytelling (descrita en el PDF) para diagramar el flujo actual de una transacción de pago internacional, identificando los actores (Cliente, Core Bancario, Fraude, Ledger) y sus interacciones dolorosas actuales.
2. Drivers de Negocio:
   - Traduzca los objetivos estratégicos en Drivers de Arquitectura concretos.
   - Especifique cómo medirá el éxito (KPIs de negocio vs. Métricas técnicas).
3. Capacidades de Negocio (TOGAF):
   - Identifique las Capacidades de Negocio (Business Capabilities) principales. ¿Cuáles deben ser optimizadas y cuáles transformadas?

Etapa 2: Diseño Estratégico.

Evaluación de: DDD, Gestión de Portafolio, Pensamiento Sistémico.

1. Descomposición del Dominio:
   - Presente un Core Domain Chart, Clasifique los subdominios en Core (Diferencial), Genérico y Soporte. Justifique por qué el "Ledger" o el "Motor de Fraude" caen en cierta categoría.
2. Bounded Contexts & Mapa de Contexto:
   - Haga diagrama de modelo de dominio
   - Defina los Bounded Contexts propuestos para la nueva arquitectura.
   - Diseñe un Context Map (Mapa de Contexto) mostrando las relaciones entre equipos y sistemas (ej. Customer-Supplier, Conformist, Anti-corruption Layer).
   - Liderazgo Técnico: Explique cómo organizará a los equipos de desarrollo basándose en la Ley de Conway Inversa para atender estos contextos.

Etapa 3: Diseño Técnico y Patrones.

Evaluación de: Arquitectura de Soluciones, Java Reactivo, Microservicios, Patrones de Integración.

1. Realizar los diagramas de arquitectura C4Model:
    - Vista de contexto
    - Visita de contenedores.
    - Vista de componentes.
2. Diagramas UML
    - Vista de despliegue
    - Vista de integración.
    - Vista de infraestructura.
3. Estilos y Patrones:
    - Seleccione y justifique los estilos de arquitectura, patrones y tácticas de arquitectura que impactan sobre los drivers de arquitectura.
    - Justifique los patrones evaluados que haya descartados y por por qué los descartó.
    - Identifique qué patrones de integración son necesarios implementar, haga un diagrama para los 3 más significativos mostrando cómo apoyan a los drivers de arquitectura.
4. Tecnología y Stack:
    - Defina el stack tecnológico basándose en Spring Boot Reactive (WebFlux). Explique por qué el modelo reactivo es superior al modelo de hilos bloqueantes para este escenario de millones de TPS.
    - Diseñe la estrategia de Consistencia de Datos. ¿Cómo manejará la consistencia eventual entre el microservicio de "Pagos" y el de "Notificaciones", versus la consistencia fuerte requerida en "Saldos"?

Etapa 4: Infraestructura y Resiliencia (Cloud & DevOps)

Evaluación de: Cloud Computing, Resiliencia, Infraestructura como Código.

1. Diseño de Despliegue:
    - Proponga una arquitectura de infraestructura en nube (AWS/Azure/GCP) que soporte Auto-scaling y Alta Disponibilidad.
    - Defina patrones de resiliencia.
2. Estrategia de Migración (Legacy a Moderno):
    - Utilizando el patrón Strangler Fig, describa una hoja de ruta (Roadmap) de 3 fases para apagar el monolito sin detener la operación.

Etapa 5: Gobierno y Liderazgo (Arquitectura Empresarial & Liderazgo)

Evaluación de: Gobierno, Análisis de Impacto, Mentoría.

1. Evaluación de Arquitectura:
    - Simule una sesión de ATAM (Architecture Tradeoff Analysis Method) ágil. Mencione 3 riesgos principales de su diseño y cómo los mitiga.

2. Gobierno de Datos y APIs:
    - Defina políticas de gobierno para la exposición de APIs.
    - Defina la estrategia de Data Governance: Calidad del dato y Linaje (Lineage) en un entorno de eventos distribuidos.

Diseño detallado

- Desarrolle los requerimientos del Anexo A.

4. Criterios de Evaluación

Su solución será evaluada bajo los siguientes pilares:

1. Coherencia Metodológica con WhyDD: ¿Siguió el flujo de Entender -> Diseñar Estratégicamente -> Diseñar Técnicamente -> Evaluar? ¿Usó los artefactos sugeridos (Domain Storytelling, Core Domain Charts)?
2. Solidez Técnica: ¿Es la solución capaz de soportar los drivers de arquitectura? ¿Es correcto el uso de programación reactiva y patrones asíncronos?
3. Visión Empresarial: ¿La arquitectura responde a los objetivos de negocio o es solo "tecnología por tecnología"?
4. Argumentación (El "Why"): ¿Justifica claramente el porqué de sus decisiones.
5. Claridad de Comunicación: Claridad y uso correcto de diagramas (C4, UML) y terminología.

Recursos Adicionales

- Se adjunta el Resumen Metodológico - Why Driven Design (PDF). Úselo como guía obligatoria para los artefactos a presentar.
- Se adjunta el ebook Modernización evolutiva de sistemas legados componibles. Úselo como guía obligatoria para los artefactos a presentar.
- Puede asumir el uso de proveedores de nube estándar (AWS, Azure) y herramientas de streaming como Kafka o Pulsar.

Anexo A: Desafío de Diseño Detallado - "El Motor de Dispersión Masiva"

Dentro de la unidad de "Global Operations", FinScale ha firmado un contrato con una Gig Economy (tipo Uber/Rappi) para procesar su nómina global diaria. Esto implica procesar un archivo batch con 500,000 instrucciones de pago simultáneas cada mañana, dirigidas a cuentas en 50 países diferentes, utilizando múltiples redes bancarias (SWIFT, SEPA, ACH Local, Cripto).

Escuela de Arquitectura Tecnológica.

Usted debe diseñar el diagrama de clases y la estructura lógica en Java para el componente MassPaymentProcessor. Debido a las restricciones de memoria y la complejidad de las reglas de negocio, se requiere explícitamente el uso de patrones de diseño para resolver los siguientes problemas técnicos:

1. Construcción de Instrucciones Complejas.

- El Problema: Cada instrucción de pago (PaymentInstruction) es un objeto complejo con más de 40 atributos (datos del beneficiario, datos fiscales, códigos de ruta, metadatos de riesgo). Muchos atributos son opcionales dependiendo del país. Además, el 90% de los pagos son recurrentes (pagos a los mismos beneficiarios semana tras semana con montos diferentes).
- Requerimiento: Diseñe un mecanismo para construir estos objetos validando su consistencia interna paso a paso.
- Optimización: Para los pagos recurrentes, el sistema carga una "Plantilla Base" del beneficiario y genera la nueva instrucción clonando la plantilla y modificando solo el monto y la fecha, evitando la re-instanciación costosa de todo el árbol de objetos.

2. Optimización de Memoria (Flyweight)

- El Problema: Al cargar 500,000 objetos PaymentInstruction en memoria para procesarlos, la JVM sufre de Out of Memory Error. Se detectó que objetos repetitivos como Currency (USD, EUR), CountryCode (US, DE) y BankRoutingInfo (Códigos SWIFT de bancos populares) se están creando miles de veces por separado.
- Requerimiento: Aplique el patrón Flyweight para que estas instancias inmutables sean compartidas entre todas las instrucciones de pago, reduciendo drásticamente la huella de memoria.

3. Pipeline de Validación de Riesgo.

El Problema: Antes de enviar un pago, este debe pasar por una serie de validaciones secuenciales que pueden cambiar dinámicamente (activarse/desactivarse en runtime):

- Validación de Sintaxis (IBAN correcto).
- Validación de Saldo (Fondos suficientes).
- Lista Negra (AML/Sanctions).
- Límites de Velocidad (No más de 5 pagos al día).
- Requerimiento: Diseñe un mecanismo desacoplado donde cada validador no conozca al siguiente, pero el pago fluya a través de ellos hasta ser aprobado o rechazado.

4. Ciclo de Vida del Pago

El Problema: El pago pasa por estados complejos: Draft -> Validated -> FX_Locked -> Sent_To_Gateway -> Clearing -> Settled (o Failed). El comportamiento del objeto cambia

según su estado (ej. no se puede llamar al método cancel() si ya está en Settled, pero sí si está en Validated). Los if-else gigantes actuales son inmanejables.

- Requerimiento: Modele el ciclo de vida utilizando el patrón State, encapsulando el comportamiento específico de cada estado en su propia clase.

5. Independencia de los canales de Pago (Bridge)

- El Problema: El sistema debe poder enviar el pago por diferentes canales técnicos sin cambiar la lógica de negocio.
- Abstracción: Transferencia Urgente vs. Transferencia Normal.
- Implementación: Red SWIFT (XML ISO 20022), Red Ripple (Blockchain API), Red Local (Archivo plano).
- Requerimiento: Desacople la abstracción del pago de su implementación técnica de envío usando algún patrón de diseño, permitiendo que una "Transferencia Urgente" pueda ser enviada vía SWIFT o Ripple indistintamente según disponibilidad.

6. Notificación Reactiva.

- El Problema: Cuando un pago cambia de estado (ej. de Sent a Failed), múltiples subsistemas dispares deben reaccionar:
- El módulo de Contabilidad debe reversar los fondos.
- El módulo de Notificaciones debe enviar un Push al usuario.
- El módulo de Analítica debe registrar el error.
- Requerimiento: Implemente el patrón Observer para que el objeto PaymentProcess notifique a estos suscriptores sin acoplarse directamente a ellos.

En su documento de diseño, dedique una sección específica a este módulo. Presente un **Diagrama de Clases UML** detallado que muestre cómo interactúan los patrones de diseño para resolver el problema del 'Motor de Dispersión'. Justifique por qué eligió cada patrón.